REMARKS

Initially, Applicant expresses appreciation to the Examiner for the courtesies extended in the recent telephonic discussion regarding this case. The remarks and amendments herein are consistent with those discussions. Accordingly, entry of this amendment and reconsideration of the pending claims is respectfully requested.

The Office Action, mailed March 16, 2007, considered and rejected claims 1-42, 45 and 46. Claims 1-42, 45 and 46 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Barnes* (U.S. Patent No. 7,096,419) in view of *Chinnici* (U.S. Publ. No. 2003/0191803), and further in view of *Ardoin* (U.S. Patent No. 6,052,691).[1]

By this paper, claims 1, 11, 20 and 29 have been amended, claims 8, 17, 27 and 34 cancelled, and no claims added. Accordingly, following this paper, claims 1-42, 45 and 46 remain pending, of which claims 1, 11, 20, 29 and 36 are the only independent claims at issue.[2]

As previously discussed with the Examiner, Applicant's invention generally relates to serializing user interface objects of custom object types and serialization formats. As reflected in claim 1, for example, an exemplary method for serializing one or more objects from an initial representation to at least one subsequent representation includes providing a serialization manager that coordinates one or more standard serialization providers identifying standard serializers, and which loads, as necessary, one or more custom serialization providers that identify custom serializers for custom object types or serialization formats. A request is then made to the serialization manager for a serializer for an object graph that comprises an object of a particular type and for a particular serialization format. Such a request is made as part of a cut, copy or paste operation and, due to the requested operation, the serializer produces a snippet of code sufficient to undo or redo a change to the object graph, and without creating a class representation of the object graph. The serializer is then called to serialize the object graph.

---

[1] As discussed with the Examiner, although pp. 4-5 of the Office Action notes that the rejection is based on only *Barnes* and *Chinnici*, the rejection further relies on *Ardoin* inasmuch as the Examiner acknowledges that *Barnes* and *Chinnici* fail to disclose each and every element of the pending claims (*see* Office Action, pp. 6-7).

Although the prior art status of the cited art is not being challenged at this time, Applicant reserves the right to challenge the prior art status of the cited art at any appropriate time, should it arise. Accordingly, any arguments and amendments made herein should not be construed as acquiescing to any prior art status of the cited art.

[2] Support for the claim amendments can be found throughout Applicant's original application, including at least the disclosure in paragraph 11 of the originally filed application, as well as in the originally filed claims.

Claims 11, 20, 29 and 36 are directed to methods and computer program products and generally correspond to the method of claim 1.

While *Barnes* and *Chinnici* generally relate to serializing objects, and *Ardoin* relates to maintaining data for allowing modifications to data to be undone or redone, the cited references, whether cited alone or in combination, fail to disclose or suggest each and every limitation of the pending claims. For example, among other things, the cited references fail to disclose or suggest wherein a serialization request is made to cause a serializer to produce a snippet of code, without creating a class representation of the object graph, as recited in combination with the other claim elements.

For example, *Barnes* generally relates to a system for serializing JavaBean state information into XML trees and structures within an application builder. (Col. 2, ll. 34-54). In particular, a frame development environment is disclosed and includes a frame that houses JavaBeans in order to provide a GUI for display and manipulation by a user. (Col. 3, ll. 23-25). As the display is manipulated, a node tree generation module is used to generate an object state node tree that captures state information of objects used within the frame. (Col. 3, ll. 32-34). With JavaBeans, a beanstate node tree is converted into an XML tree used by an XML tree generation module, and each beanstate node writes itself as XML. (Col. 3, ll. 51-54). A resulting file may then be created that contains the object state information of the JavaBean in an XML tagged format that fully specifies the public and private states of the objects in the frame. (Col. 3, ll. 62).

Chinnici generally relates to deserialization of objects and, in particular, of supporting extensible type mappings between XML data types and Java types, in which there is a Java type serializer and an XML type deserializer. (¶¶ 126, 128). The extensible type mappings may be used by an API to allow the format of data to be changed from XML to Java based on the type mapping defined between the corresponding Java type and the XML data type, as it is moved from a client to server. (¶¶ 126-129). For instance, a serializer interface may include a serializer for Java arrays and classes and use an XML schema definition to generate a corresponding XML representation from the Java object. (¶ 129).

Ardoin relates to a method for maintaining relationships between entities in a computer system, and particularly to CAD/CAM software systems for managing objects so that when an object is modified, copied, or deleted, referential integrity is maintained, thereby allowing

undoing or reversing actions taken on the object. (Col. 1, ll. 32-43; Col. 3, ll. 13-29). More particularly, *Ardoin* describes a system which includes a relations subsystem that describes the associative and constrain relationships between various objects by using an associative graph. (Col. 6, ll. 28-35). Within the associative graph are nodes that are connected and represent related entities, and edges between the nodes, which represent the relationships between the entities. (Col. 6, ll. 28-44). The graph itself may be modified when nodes/entities are modified, copied, or deleted. (Col. 6, ll. 35-39). Further, the system of Ardoin can include an optional undo interface. When objects are modified, the undo interface writes modifications to a log, which log can thereafter be read to restore a value. (Col. 41, ll. 47-54).

Accordingly, the cited references generally teach that data may be serialized from Java to XML to create an XML document representation of the data (*Barnes* and *Chinnici*) and that when a node is modified, copied or deleted, the node is so modified to write modifications to a log file for later restoring a value, if needed (*Ardoin*). Notably, however, none of the references teach that the created XML document or the generated log file, is "without...a class representation of the object graph" as recited in combination with the other claim elements.

For this teaching the Office Action recites *Barnes*, with regard to its production of XML "rather than a class representation of the object graph" (Office Action, p. 9) and summarily concludes that an "XML representation of the object graph does not constitute a class representation of the object graph." (Office Action, p. 4). Applicant respectfully submits that this conclusion is without support in any of the cited references. Thus, the Office Action appears to argue, either based on an argument of inherency or based on the Examiner's own knowledge, that an XML document is not an class representation. Nevertheless, whether even were the XML document or the log file itself not considered a class representation, the assertion in the Office Action appears to fail to consider that if the XML document or log file includes a class representation of the object graph, it also fails to disclose generating a snippet of code "without...a class representation of the object graph" as recited.

While the Office Action concludes that an XML representation is not itself a class representation, and even where this accepted as true, the cited references fail to have any disclosure that when an XML file or log file is created, there are no class representations produced within the XML representation or log file. In fact, *Chinnici* expressly notes that when

a Java class is serialized, a corresponding XML representation is created, thus suggesting that the XML representation includes an XML class corresponding to the Java class. (¶ 129).

Further, a conclusion that XML representations do not include class representations is equally incorrect. As merely one example, U.S. Patent No. 7,020,641, issued to *Leong* and entitled "Method, System, and Program for Maintaining a Database of Data Objects," describes a system similar to the cited references in which data objects in a database are in one programming language and are converted to a second programming language. (Col. 2, ll. 15-25). As disclosed in *Leong*, a tagged XML file expressly includes classes and can be specifically employed to translate between object types. (*See, e.g.*, Figs. 2, 3b, 4; Col. 4, ll. 1-27; Col. 4, ln. 63 to Col. 5, ln. 9). Thus, *Leong* expressly describes a tagged XML document similar to the tagged XML document in *Barnes*, and that the tagged XML document does include class representations of data. Accordingly, any assertion that an XML document cannot include class representations is clearly in error.

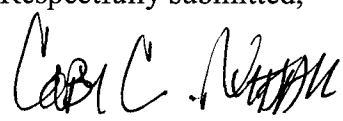
Inasmuch as a *prima facie* case of obviousness requires, among other things, that the cited references teach or suggest all the claim limitations, Applicant respectfully submits that the rejections of record are overcome. (*See* M.P.E.P. § 2143). Specifically, the cited references are silent as to whether an XML document or log file includes therein a class representation of an object graph or user interface and, if anything, *Chinnici* suggests that a generated XML file does include a class representation of a Java class. Further, XML documents are not necessarily without class representations as evidenced by *Leong*.

In view of the foregoing, Applicant respectfully submits that the other rejections to the claims are now moot and do not, therefore, need to be addressed individually at this time. It will be appreciated, however, that this should not be construed as Applicant acquiescing to any of the purported teachings or assertions made in the last action regarding the cited art or the pending application, including any Official Notice. Instead, Applicant reserves the right to challenge any of the purported teachings or assertions made in the last action at any appropriate time in the future, should the need arise. Furthermore, to the extent that the Examiner has relied on any Official Notice or the Examiner's own knowledge, explicitly or implicitly, Applicant specifically requests that the Examiner provide references supporting the teachings officially noticed, as well as the required motivation or suggestion to combine the relied upon notice with the other art of record or a supporting affidavit, as relevant.

In the event that the Examiner finds remaining impediment to a prompt allowance of this application that may be clarified through a telephone interview, the Examiner is requested to contact the undersigned attorney by telephone at (801) 533-9800.

Dated this 14th day of May, 2007.

Respectfully submitted,



RICK D. NYDEGGER
Registration No. 28,651
JENS C. JENKINS
Registration No. 44,803
COLBY C. NUTTALL
Registration No. 58,146
Attorneys for Applicant
Customer No. 047973

RDN:JCJ:CCN:gd
GD0000001796V001